

Express Mail No. EL 727968110US

Date of Deposit: March 23, 2001

APPLICATION FOR LETTERS PATENT  
OF THE UNITED STATES

NAME OF INVENTORS: GEORG MUENZEL  
48 Parker Road  
Plainsboro, New Jersey 08536

TITLE OF INVENTION: INDUSTRIAL AUTOMATION SYSTEM  
GRAPHICAL PROGRAMMING LANGUAGE  
STORAGE AND TRANSMISSION

TO WHOM IT MAY CONCERN, THE FOLLOWING IS  
A SPECIFICATION OF THE AFORESAID INVENTION

## TITLE OF INVENTION

### INDUSTRIAL AUTOMATION SYSTEM GRAPHICAL PROGRAMMING LANGUAGE STORAGE AND TRANSMISSION

5

## CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims benefit from U.S. Provisional Patent  
Application No. 60/192,147, filed March 24, 2000, under 35 U.S.C. § 119(e).

10

## FIELD OF THE INVENTION

The present invention relates generally to graphical programming  
languages for programmable logic controllers. In particular, the invention  
concerns a method and system for standardized storage of graphical  
programming languages.

15

## BACKGROUND

Graphical programming languages are widely used in the field of  
industrial automation. They provide an intuitive way for automation engineers  
to specify the control logic for an industrial control application to be run by a  
controller, usually a programmable logic controller ("PLC"). A PLC may  
comprise dedicated hardware or, alternatively, be implemented in software on  
a conventional personal computer, the latter being sometimes referred to as a  
PC-based PLC. The term PLC will be used here to describe either type of  
industrial controller.

20

25

Existing graphical programming systems for industrial automation  
control software typically provide a graphical editor that embodies features

that are well-known in the context of text editing. Using a system of this kind, an automation engineer interacts with an editor to select icons from a menu in such a manner as to structure the control flow for the controlled industrial process, set conditions to be observed in that control, and so forth. The

5 symbols available for use via the editor correspond to the particular graphical programming language being used, among which languages are: ladder logic, function block diagrams, sequential function charts and flowcharts, and languages if any embodying other formalisms. The graphical symbols depicted for the engineer by these editors are represented, when stored on a

10 hard drive, for example, by the computer system on which the editor runs, in a private or internal binary form, here referred to as an "internal representation", which is essentially a set of software objects that use volatile memory (RAM) (hereinafter referred to as "memory" or "computer memory") and have associated code. This internal representation is specific or private

15 to the software vendor, rather than being standardized.

When an industrial control program is deemed sufficiently complete to be debugged, or to be run on the PLC, the system compiles the internal representation to arrive at another binary form that is more readily usable by the PLC. In some systems, this compilation step is direct; in others, several

20 layers of compilation are used for reasons unrelated to the present invention.

Some of the graphical programming languages in use today are the subject of international standards, such as are defined in IEC 61131. In contrast to textual programming languages, however, which can be stored in a computer file exactly as the user typed them (i.e., in a serialized form),

there is no commonly agreed upon storage format for graphical programming languages. The representations used in existing graphical language programming systems for industrial control applications, moreover, are not generally human-readable. Nor are they available in a format capable of being interpreted by a browser, such as Internet Explorer 5, or one that is easily or quickly parsed.

The known ways of attempting to address these shortcomings have involved the use of one or another binary format, which has the disadvantage of being private and unreadable with a standard word-processor.

Alternatively, a proprietary text format, while capable of being more readable, must be fully defined. That is, it must be shown to follow the rules of a programming language ("grammar"). In order to understand such a program after reading it from a file, a full-blown parser must be written. These shortcomings have limited the utility of programs created using graphical programming systems and placed constraints on the process of developing control programs.

## SUMMARY OF THE INVENTION

The present invention is directed at overcoming the shortcomings of existing industrial automation graphical programming systems described above by providing methods and computer program products for storing graphical, industrial automation programs in a standard format, one that is serialized, relies on a text-based language (i.e., a mark-up language),

includes tags or analogous functionality for identifying items, and that has as the ability to describe data hierarchically. More specifically, the present invention provides a mechanism that is standardized, readable by a human, supported by existing browser technology (e.g, Microsoft Internet Explorer 5 ("IE5")), is easy and fast parsing, and that supports hierarchical information structures.

The present invention also provides methods, systems and computer program products that permit industrial automation control programs, once created in whole or in part, to be transmitted over a network in an easily- displayed and apprehended form. The program code stored in this standard, readable form can be transmitted over a network to, or received from, a plurality of computer systems. In addition, markup language schemas (or analogous definitions) describing content models for markup language files generated by graphical programming language applications can be made available to a plurality of developers by posting, for example, on an internet site. This approach is intended to permit, among other advantages, distributed generation of industrial automation program code or applications.

In addition, or alternatively, code generated by a first system employing a first internal representation of code generated by a graphical programming language can be converted to the markup-language (e.g., XML) format, transmitted to a second system employing a second internal representation of the code, and there be reconverted to the second internal representation. The present invention, in this embodiment, is thereby capable of providing interoperability between systems.

Accordingly, an embodiment of the present invention provides a method for representing industrial automation computer program code created using a graphical programming language tool that stores the created code in computer memory in an internal representation during execution.

- 5 The method comprises the steps of identifying industrial automation code in computer memory in the internal representation and converting the code from the internal representation to a markup language format.

- Another embodiment of the present invention involves a computer program product used in conjunction with a computing device for creating  
10 industrial automation system control program code with a graphical language programming tool and storing the code in a computer memory in an internal representation during execution. The computer program product comprises a computer usable medium comprising computer readable program code for identifying industrial automation system control program code stored in  
15 computer memory in the internal representation. The computer program product further comprises computer readable program code for converting the identified industrial automation control program code from the internal representation to a markup language format.

- A further embodiment of the present invention involves a computer  
20 program product that comprises a computer-readable storage medium and has data stored on it that comprises a representation of industrial automation control code formatted in markup language.

Another embodiment of the present invention relates to a computer program product for permitting a user to create industrial automation control

programs. The product comprises a computer-readable storage medium having computer program code stored on it. The computer program code comprises industrial automation graphical programming language code. The graphical programming language code comprises an editor adapted to permit  
5 the user to create industrial automation control code using graphical elements, the control code being stored in memory in an internal representation during execution; and computer program code for converting industrial automation control code, stored in memory in the internal representation, from the internal representation to a markup language format.

10 In another embodiment of the present invention, a method is provided for communicating the logical structure of industrial automation control program data to permit a plurality of application developers to create applications relating to the data. The method comprises the steps of creating a schema defining a content model for markup language files generated by  
15 an industrial automation control program system and posting the schema for access over a network by the application developers.

Still further, an embodiment of the present invention entails a method for providing industrial automation control code from a server system over a network to which the server system is coupled and to a client system also  
20 coupled to the network. The method comprises the steps of accessing a markup-formatted version of the control code and transmitting the accessed, markup-formatted control code over the network in connection with a network address corresponding to the client system, thereby causing the transmitted, markup-formatted control code to be received by the client system.

Yet another embodiment of the present invention relates to a method for programming industrial automation control applications comprising the steps of providing a computer system coupled to a network, configuring the first computer system to receive over the network transmissions of data from a plurality of industrial automation program developer systems, and receiving data from the plurality of industrial automation program developer systems program code in a markup language format.

#### BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 provides, in schematic form, an illustration of an embodiment of the computer program product according to the present invention in the context of an industrial automation control system that includes an industrial automation control programming system.

Figure 2 provides, in schematic form, an illustration of an embodiment of a conversion process according to the present invention.

Figure 3 provides an illustration of an object model for an internal representation of a flowchart which, according to an embodiment of the present invention, is to be converted into a markup format.

Figure 4 provides an illustration of an object model for an internal representation of a flowchart body (corresponding to the flowchart object model of Figure 3) which, according to an embodiment of the present invention, is to be converted into a markup format.

Figure 5 provides an illustration of an object model for an internal representation of a flowchart interface (corresponding to the flowchart object

model of Figure 3) which, according to an embodiment of the present invention, is to be converted into a markup format.

Figure 6 provides an illustration of an embodiment of a system for deploying computer program product according to the present invention and  
5 for performing an embodiment of one or more methods according to the invention.

### DETAILED DESCRIPTION OF THE INVENTION

The various embodiments of the invention briefly described above,  
10 and set forth in the appended claims, are described below with reference to the figures, as well as to the code provided at the end of the text.

The present invention is directed to the creation of a standard, human-readable, preferably browser-readable representation of otherwise non-standardized representations of graphical programming language code  
15 for industrial automation. In a presently preferred embodiment of an aspect of the invention, XML is used as a standard storage format. XML, short for "the Extensible Markup Language", is a subset of the Standard Generalized Markup Language ("SGML") and is, essentially, a set of rules for defining a text based markup language. See, for example, *XML IE5 Programmer's*

20 *Reference*, by A. Homer, WROX Press Ltd., 1999 and *Applied XML: A Toolkit for Programmers*, by A. Cekonkus and F. Hoodbhoy, John Wiley & Sons, Inc., 1999, the contents of which are herein incorporated by reference in their entirety. The invention is not limited to the use of XML, but can also

be embodied with other markup languages corresponding to the definition set forth below

Moreover, the present invention can be practiced using Microsoft Visual Studio 6.0, as well as Microsoft XML (available as part of Internet Explorer 5).

For each graphical language used in the field of industrial automation, a set of XML tags, elements and attributes, as well as an XML schema (or document type definition "DTD") are defined. A specific computer program, an example of which is described below for the conversion of a flowchart program to XML, is used to transform, convert or serialize the graphical program.

A number of terms frequently used in this document are defined below.

The term "data storage device," as used here, refers to any medium for the computer-retrievable storage of data including, without limitation, memory (e.g., RAM), hard disk, compact disk, floppy disk, or other storage device.

The term "computer program product", as used here, includes any product capable of retaining data that may include computer program code and that can be permanently or temporarily coupled to a computer system that can retrieve data from the computer program product. Computer program products include media that are sold to users of computer systems so that the computer systems can operate in accordance with content stored on them. The term also encompasses hardware coupled to a computer

system onto which content has been downloaded, for example, over a network, so that the computer system can operate in accordance with that content.

The term “editor command”, as used here, encompasses any  
5 command typically associated with known editors and involving the manipulation of text, code or the like, the commands including, for example, cut, copy, paste, move, delete, save, save as, undo, redo, and so forth.

The term “graphical programming language”, as used here,  
includes ladder logic, function block diagrams, sequential function charts and  
10 flowcharts and other graphical languages, whether now in existence or yet to be developed.

The term “markup-formatted”, as used here, refers to the state of having been stored in a markup language format or having been converted (e.g., from a graphical programming language internal representation) to a  
15 markup language format (markup being used in the sense defined above).

The term “markup language”, as used here, refers to text-based mark-up languages including but not limited to those that are subsets of the Standard Generalized Markup Language, SGML, which use elements that comprise a string of characters, including an individual character string that  
20 defines the opening or closing part of the element (corresponding to the term “tag” in XML usage), a name and value pair enclosed within the element’s opening character string or tag, the element attribute names and their values, the content of the element and any closing tag, a character string that defines the opening or closing part of an element

The term "network" refers, in a preferred embodiment of the invention, to an internet, but also encompasses any type of data communication network, whether wired or wireless.

An embodiment of the computer program product according to the present invention is shown in schematic form in Figure 1. In that figure, the computer program product is depicted in the context of an industrial automation control system, including an industrial automation control programming system 10, an industrial controller system 20 and a controlled process 30. Industrial controller system 20 may be a PLC that is separate hardware from the computer on which the programming system 10 runs; alternatively, industrial controller system 20 and programming system 10 could be implemented on the same computer device (e.g., embodying what is often referred to as a "PC-based PLC"). The typical programming system, which nowadays allows an industrial automation engineer to program with graphical tools (flowchart elements, for one of several examples), includes an editor 12. Editor 12, when operated by an automation engineer, graphically displays, in whatever formalism it uses, the program created by the engineer. At the same time, it causes the creation and storage in a computer memory of an internal representation (as elaborated upon in Figure 2 and the accompanying text).

The control programming system 10 (one example of which is Step 7®, developed and marketed by Siemens A.G. and Siemens Energy & Automation, Inc.) also may include one or more compilers 14, which convert, either directly or indirectly, the internal representation created using

the editor 12 into a form that is understandable by the controller 22 of industrial controller system 20. Using the compiled result, and based also on clock data (not shown) and on input received from controlled process 20, controller 22 generates control instructions for running process 30. In addition, the compiled code understandable by controller 22 can be stored on data storage device 26, that is coupled to (or is part of) industrial controller 20.

Another component of programming system 10, according to an embodiment of the present invention, is a converter 16 for converting the internal representation of control programs generated by editor 12 to a markup language format (e.g., XML). The operation of converter 16 is elaborated upon below in connection with Figure 2 and in the appended source code. The markup language formatted code generated by converter 16 can be stored either on data storage device 26, with assistance of identification and location program code 24 running on industrial controller 20, or, alternatively, can be transmitted to network 40 and, via that network, to other systems (not shown).

Figure 2 provides, in schematic form, an illustration of the steps according to an embodiment of a method 50 according to the present invention. A sample of a flowchart program 52 (e.g., generated by editor 12 of Figure 1) is given an internal representation 60 that is usually in binary format, which is held in memory (RAM) (not shown) during execution of the program. The internal representation 60 is, in general, specific to the vendor

of the graphical programming language system 10, is not readable by a human, is not readable using a word-processor, nor using a browser.

The internal representation is converted (or "serialized") into the format of a suitable markup language (as set forth in the corresponding definition, above). Once converted, the graphical program is available in a markup-formatted form 64 (an example of which embodying XML is shown in Figure 2) and can be stored (e.g., in data storage device 26 of Figure 1 and Figure 6). This markup-formatted form 64 of the graphical programming language code, originally represented at 52, can be sent directly to a monitor or display 28, where it can be viewed with known viewing software, including word processing or browser software. It can also be sent to printer 68, to create a human-readable hardcopy. Alternatively, it could be sent over a network 40 to another computer 70, which may have an associated interface 72. Computer 70 could be devoted, for example, to permitting development of control programs, which can then be converted and transmitted or re-transmitted (although not necessarily in that order) to an industrial controller 20, programmed using graphical programming language system 10, where it can then be deployed.

When it becomes necessary to edit or compile an industrial automation program code that is already in markup format, at reference numeral 64, the markup-formatted code 64 is converted back (or "deserialized") from markup language representation to the internal representation 60 (see, e.g., source code appended below).

The steps of the method of Figure 2 may be invoked any time it is convenient or necessary to store or view, or to transmit to others for storage or viewing, a graphical industrial control program in a standardized representation. For example, any time an item is selected using an editing  
 5 function, such as drag and drop, copy, cut, paste, undo, redo, etc., the conversion can be performed, creating a markup language (e.g., XML) string in memory that can be placed, for example, on a clipboard for transfer elsewhere. If a "save" were to be done to a graphical industrial automation program, or part of one, it would be converted, at 62, to markup format (e.g.,  
 10 XML) and saved in a file, for example, on storage device 26 of Figure 1. Upon file "open" command being invoked relative to that stored, markup-formatted file, the file would be read and converted, at 66, back to the internal representation.

Figure 3 provides an illustration of an object model for an internal  
 15 representation of a graphical programming language formalism. As in Figure 2, reference numeral 52, a flowchart formalism, is used for purposes of illustration. The corresponding internal representation, reference numeral 60 in Figure 2, is to be converted into a markup format. This object model, of flowchart type (FChType) may, like the other object models, be implemented  
 20 using COM ("Common Object Model") technology, available from Microsoft Corp., or other suitable tools (See Class FChType, in the appended source code, below). Object FChType includes within its structure a flowchart body object, FchBody, and an interface object, FchInterface, both in a one-to-one aggregation relationship with object FChType. (See legend in Figure 3).

Figure 4 provides an illustration of an object model for an embodiment of the present invention, specifically focusing on an object model of a flowchart body, FChBody corresponding to the object model illustrated in Figure 3. Body object FChBody stands in a one-to-one aggregation

- 5 relationship to a flowchart elements object, FChElements, as well as with a flowchart links element, FChLinks, the latter being in a one-to-many aggregation relationship with a flowchart link element, FChLink. FChElements, in turn, stands in a one-to-many aggregation relationship with one or more FChElement instances, each of which is related FChLink. A
- 10 FChLink object connects 2 FChElement objects, a SourceElement to a TargetElement.

- Each FChElement stands in a one-to-one aggregation relationship with an FChInstance object, which in turn stands in a one-to-one aggregation relationship with a FChAssignments object. Each FChAssignments object
- 15 stands, in turn, in a one-to-many aggregation relationship with one or more FChAssignment objects.

- Figure 5 provides an illustration of an object model for an embodiment of the present invention, specifically focusing on an object model of a flowchart interface, corresponding to the object model illustrated in Figure 3.
- 20 The FChInterface object stands in a one-to-one aggregation relationship with FChInterfaceltems object, and in a one-to-many relationship with the FChInterfaceltem. Moreover, FChInterfaceltems object is in a one-to-many FChInterfaceltem object.

Referring again to Figure 2, the internal representation 60, described above in connection with Figures 3, 4 and 5, is converted at reference numeral 62 to a suitable markup language format, for example XML. See the commented source code, below, for further detail.

5 Figure 6 provides an illustration of an embodiment of a system for deploying computer program product according to the present invention and for performing an embodiment of one or more methods according to the present invention. An industrial automation programming and control system 18, which can include or incorporate a PLC 20 (as shown by the dotted lines)  
10 is coupled to a display 28, to at least one data storage device 26 and to a controlled process 30. In addition, it is coupled to a network 40, over which it can communicate with other computers also connected directly or indirectly to the same network 40. For example, industrial automation programming and control system 18 can be in communication over network 40 with a remote  
15 computer 70 having a display 72 and data storage device(s) 74, or with a plurality of such computers, one of which is shown at reference numeral 80, also having a display 82 and data storage device(s) 84.

By using the conversion approach shown in Figure 2 and described in the accompanying text, not only can markup-formatted code be easily viewed  
20 at the site where it was created, but can easily be sent over a network 40 to another computer 70, where an operator may, using display 72, readily examine the code on the display, using a browser, for example. If the operator were an industrial automation controls engineer or developer of industrial automation control code, that operator could generate program

code on computer 70 that could subsequently be converted to markup format and transmitted or re-transmitted (although not necessarily in that order) to an industrial automation programming and control system 18 or controller 20. The same could be done using computer 80, or via any number of computers  
5 in communication over network 40 with automation programming and control system 18.

Communications over network 40, preferably although not necessarily an internet, between various involved computers depicted in Figure 6 can be done in any suitable manner including, without limitation, via downloading of  
10 pages using hypertext transfer protocol, or even via sending electronic mail messages.

Given this configuration, in an embodiment of an aspect of the present invention, computer 70 could be considered an industrial automation control code server system coupled over a network to a client system 18. Computer  
15 70 accesses a markup-formatted version of the control code, transmits the accessed, markup-formatted control code over the network in connection with a network address corresponding to system 18, thereby causing the transmitted, markup-formatted control code to be received by the client system. Moreover, system 18, in response to the received markup-formatted  
20 control code, may transmit to computer 70 over the network 40 data relating to the automation to which the markup-formatted control code is directed. Furthermore, computer 70 can generate or otherwise access control code modified in response to receipt of the data from system 18, wherein the modified control code is markup-formatted. In addition, the markup-

formatted, modified control code can be transmitted over the network in connection with a network address corresponding to the system 18, thereby causing the transmitted, modified, markup-formatted control code to be received by the system 18.

5           Figure 6 depicts an embodiment of another aspect of the present invention involving a method for communicating the logical structure of industrial automation control program data to permit a plurality of application developers to create applications relating to the data. According to the method, a schema (or analogous data) (see source code for an example  
10   schema appended below) defining a content model for markup language files generated by an industrial automation control program system (e.g., XML) is posted for access over network 40 (e.g., internet). Application developers using, for example, computers 70, 80 and 90, can then access and understand the logical structure of the graphical programming language data  
15   and can write their own applications. Developers and systems that communicate with one another using the standardized format according to the present invention need not use identical internal representations 60 of the automation system control code, provided that their conversion program takes into account the particulars of the internal representations 60 they do  
20   use.

Figure 6 also describes a system in which a method for providing industrial automation control code services can be implemented. Assuming computer 70 can be considered a server running software permitting the creation of markup-formatted industrial automation control code (e.g.,

reference numeral 62 of Figure 2), computer 70 can access such a markup-formatted version of the control code and transmitting the accessed, markup-formatted control code over the network 40 to a client system, for example, computer 18 in connection with a network address corresponding to computer  
 5 18, thereby causing the transmitted, markup-formatted control code to be received by the client system 18.

Client system 18, which (possibly along with PLC 20), controls process 30, may, in response to receiving the markup-formatted control code (e.g., reference numeral 62), may transmit to the server system 70 data relating to  
 10 the automation to which the markup-formatted control code is directed.

Server system 70 may modify code it is generating or has generated and, where it has access to automation system control code modified in response to receipt of system data from the client system 18, it may transmit the markup-formatted, modified control code over the network in connection with  
 15 a network address corresponding to the client system 18, thereby causing the transmitted, modified, markup-formatted control code to be received by client system 18.

In another embodiment of the present invention, the foregoing method may involve a second client system 90 coupled to the network. Server 70  
 20 would transmit the accessed, markup-formatted control code (62, Figure 2) over network 40 in connection with a network address corresponding to the second client system 90, thereby causing the transmitted, markup-formatted control code to be received by the second client system 90.

In yet another embodiment of the present invention, which

demonstrates the potential for increased interoperability of systems, the first client system 18 may be configured to reconvert the markup-formatted control code to a first internal representation, while the second client system 96 is configured to reconvert the markup-formatted control code to a second  
5 internal representation.

Finally, Figure 6 also is directed to a method for programming industrial automation control applications using a plurality of distributed applications developers. A computer system 18 is provided and coupled to a network 40 and configured to receive over the network 40 transmissions of  
10 data from a plurality of industrial automation program developer systems 70,...,80, the transmissions comprising data from program developer systems 70,...,80, in a markup language format.

In addition to the embodiments of the aspects of the present invention described above and in the XML schema and source code listings set forth  
15 below, those of skill in the art will be able to arrive at a variety of other arrangements and steps which, if not explicitly described in this document, nevertheless embody the principles of the invention and fall within the scope of the appended claims.

**Example of a Flowchart program in XML:**

```

<?xml version="1.0" ?>
<ChartSource ExecutableTypeName="FUNCTION_BLOCK">
  <Interface>
5    <Section Name="VAR_INPUT">
      <Item Name="Enabled" Mode="ReadOnly" Type="Bool" Init="TRUE"/>
      <Item Name="Frozen" Mode="ReadOnly" Type="Bool" Init="FALSE"/>
    </Section>
    <Section Name="VAR_OUTPUT">
10   <Item Name="Active" Mode="ReadOnly" Type="Bool" Init="FALSE"/>
    </Section>
    <Section Name="VAR">
      <Item Name="Index" Mode="ReadOnly" Type="Int" Init="0"/>
      <Item Name="Internal" Mode="ReadOnly" Type="Struct">
15   <Item Name="Trace" Mode="ReadOnly" Type="Array [1..32] of Bool"/>
      <Item Name="CurrentStep" Mode="ReadOnly" Type="Int" Init="0"/>
    </Item>
      <Item Name="xxx" Mode="ReadOnly" FChType="Called" Type="FB16"/>
    </Section>
20  </Interface>
    <Body>
      <Elements>
        <Element Number="0" Type="TBegin" Caption="Begin"/>
        <Element Number="1" Type="TEnd" Caption="End"/>
25   <Element Number="2" Type="TSubChart" Caption="Test1">
      <Instance Name=" Test1" InterfaceVersion="14" ChartType="Called">
        <Assignment Name="BoolPara" Value=""/>
        <Assignment Name="IntPara" Value=""/>
      </Instance>
30  </Element>
    </Elements>

```

```

<Links>
<Link Number="1" SourceElement="2" TargetElement="1" Index="0"
Caption=""/>
<Link Number="2" SourceElement="0" TargetElement="2" Index="0"
5  Caption=""/>
</Links>
</Body>
</ChartSource>

```

10           The XML string can be validated using an XML-Schema. A schema describes the elements and attribute allowed in an XML file.

Example: Schema file for a flowchart

```

<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">

```

15

```

<ElementType name="Comment" content="textOnly"/>

```

```

<AttributeType name="Name" dt:type="string"/>

```

```

<AttributeType name="Value" dt:type="string"/>

```

20

```

<ElementType name="Attribute" content="empty">

```

```

  <attribute type="Name" required="yes"/>

```

```

  <attribute type="Value" required="yes"/>

```

```

</ElementType>

```

25

```

<ElementType name="Attributes" content="eltOnly">

```

```

  <element type="Attribute" maxOccurs="*/>

```

```

</ElementType>

```

30

```

<AttributeType name="Mode" dt:type="enumeration" dt:values="Mixed
ReadOnly Edit"/>

```

```

<AttributeType name="Type" dt:type="string"/>

```

<AttributeType name="Init" dt:type="string"/>

<ElementType name="Item" content="eltOnly">

<attribute type="Name" required="yes"/>

5 <attribute type="Mode" required="yes"/>

<attribute type="Type" required="yes"/>

<attribute type="Init"/>

<element type="Comment" minOccurs="0"/>

<element type="Attributes" minOccurs="0"/>

10 <element type="Item" minOccurs="0" maxOccurs="\*/"/>

</ElementType>

<ElementType name="Section" content="eltOnly">

<AttributeType name="Name" dt:type="enumeration"

15 dt:values="VAR\_INPUT VAR\_OUTPUT VAR\_IN\_OUT VAR VAR\_TEMP"/>

<attribute type="Name" required="yes"/>

<element type="Item" minOccurs="0" maxOccurs="\*/"/>

</ElementType>

20 <ElementType name="Interface" content="eltOnly">

<element type="Section" maxOccurs="\*/"/>

</ElementType>

<ElementType name="Assignment" content="empty">

25 <attribute type="Name" required="yes"/>

<attribute type="Value" required="yes"/>

</ElementType>

<AttributeType name="ChartType" dt:type="string"/>

30

<ElementType name="Instance" content="eltOnly">

<attribute type="Name" required="yes"/>

```
<attribute type="ChartType" required="yes"/>
<element type="Assignment" maxOccurs="*/>
</ElementType>
```

5 <ElementType name="SourceCode" content="textOnly"/>

```
<AttributeType name="Number" dt:type="int"/>
<AttributeType name="Caption" dt:type="string" default=""/>
```

10 <ElementType name="Element" content="eltOnly">

```
<attribute type="Number" required="yes"/>
<attribute type="Type" required="yes"/>
<attribute type="Caption" required="no"/>
<group minOccurs="0" order="one">
```

15 <group order="seq">

```
<element type="Comment"/>
<element type="SourceCode"/>
</group>
<element type="Instance" maxOccurs="*/>
```

20 </group>

```
</ElementType>
```

```
<ElementType name="Elements" content="eltOnly">
```

```
<element type="Element" maxOccurs="*/>
```

25 </ElementType>

```
<AttributeType name="SourceElement" dt:type="int"/>
```

```
<AttributeType name="TargetElement" dt:type="int"/>
```

```
<AttributeType name="Index" dt:type="int"/>
```

30

```
<ElementType name="Link" content="empty">
```

```
<attribute type="Number" required="yes"/>
```

```
<attribute type="SourceElement" required="yes"/>
```

```
<attribute type="TargetElement" required="yes"/>
```

```
<attribute type="Index" required="yes"/>
```

```
<attribute type="Caption"/>
```

```
5 </ElementType>
```

```
<ElementType name="Links" content="eltOnly">
```

```
<element type="Link" maxOccurs="*/>
```

```
</ElementType>
```

```
10
```

```
<ElementType name="Body" content="eltOnly" order="seq">
```

```
<element type="Elements"/>
```

```
<element type="Links"/>
```

```
</ElementType>
```

```
15
```

```
<AttributeType name="ExecutableTypeName" dt:type="string"/>
```

```
<ElementType name="ChartSource" content="eltOnly" order="seq">
```

```
<attribute type="ExecutableTypeName" required="yes"/>
```

```
20 <element type="Interface"/>
```

```
<element type="Body"/>
```

```
</ElementType>
```

```
</Schema>
```

```
25
```

This embodiment of a schema describes the content model for XML files generated by converting a flow chart application. Such a schema can be posted, over a network, for example, so that other users can understand the logical structure of the flow chart data and thereupon write applications manipulating this data in a way they see fit. Like Document Type Definitions

```
30
```

(DTD's), which might be considered their predecessors, schemas provide a way of describing the structure of XML data. Schemas may be preferable to DTDs, in that, unlike DTDs, they use a syntax to that of XML. Also, they allow a more precise description than do DTDs, because they incorporate

5 data typing and inheritance.

The following is, for an embodiment of the present invention, a description of the requirements a document must meet to be validated by flwschma.xml, starting with the root element <ChartSource>:

(Note: All attributes are of type 'string' unless otherwise stated.)

- 10 > **<ChartSource>** - The schema's root element, <ChartSource> requires an "ExecutableTypeName" attribute, one <Interface> element, and one <Body> element.
  - > **<Interface>** - This element requires at least one <Section> element.
  - > **<Section>** - This element requires a "Name" attribute. "Name"
 15 must have one of the following values: VAR\_INPUT, VAR\_OUTPUT, VAR\_IN\_OUT, VAR, VAR\_TEMP. There may be any number of <Item> elements.
    - > **<Item>** - This element requires "Name," "Mode," and "Type" attributes. "Mode" must have one of the following values:
 20 Mixed, ReadOnly, Edit. The following are optional: an "Init" attribute, one <Comment> element, one <Attributes> element, and any number of <Item> elements.
      - > **<Comment>** - This element contains text only.
      - > **<Attributes>** - This element requires at least one
 25 <Attribute> element.
        - > **<Attribute>** - This element requires "Name" and "Value" attributes.

- **<Body>** - This element requires one <Elements> element and one <Links> element.
- **<Elements>** - This element requires at least one <Element> element.
- 5       ➤ **<Element>** - This element requires "Number" and "Type" attributes. "Number" is of type 'int.' The following are optional: a "Caption" attribute, and either 1)the sequence of one <Comment> element followed by one <SourceCode> element or 2)any number of <Instance> elements.
- 10       ➤ **<Comment>** - This element contains text only.
- **<SourceCode>** - This element contains text only.
- **<Instance>** - This element requires "Name" and "ChartType" attributes. Optional is any number of <Assignment> elements.
- 15       ➤ **<Assignment>** - This element requires "Name" and "Value" attributes.
- **<Links>** - This element requires at least one <Link> element.
- **<Link>** - This element requires "Number," "SourceElement," "TargetElement," and "Index" attributes. "Number," "SourceElement," "TargetElement," and "Index" are of type 'int.' Optional is a "Caption" attribute.
- 20

Source Code

## 25 **Class FChType**

\*\*\*\*\*

!\* Creates a FChType object from a XML string

!\* NewContents contains the XML string

\*\*\*\*\*

30

On Error GoTo ErrorHandler

Dim strContents As String

Dim regserv As FChRegServer.FChRegistry

```

Dim strTemplateFolder As String
Dim xmlns As String
Dim FileName As String
Dim fso As Scripting.FileSystemObject
5   strContents = NewContents
   If Mid$(strContents, 1, 1) = ChrW$(&HFEFF) Then
       strContents = Mid$(strContents, 2)
   End If
   Set fso = New Scripting.FileSystemObject
10  Set regserv = New FChRegServer.FChRegistry
   strTemplateFolder = regserv.GetTemplateFolderPath
   Set regserv = Nothing
   FileName = strTemplateFolder & "fbschema.xml"
   If fso.FileExists(FileName) Then
15     xmlns = "<ChartSource xmlns='x-schema:" & FileName & "' "
       strContents = Replace$(strContents, "<ChartSource ", xmlns, 1, 1,
vbTextCompare)
   End If
   XMLReadProperties strContents
20  Exit Property
ErrorHandler:
   ErrorMessage Err.Number, "FChType.Contents", UNEXPECTED
End Property

25
*****

Public Property Get Contents() As Variant
*****

'* Gets the contents of a FChType object as XML string
30 '* Return value is the XML string representing this object.
*****

On Error GoTo ErrorHandler

```

```

Dim strContents As String
Dim b() As Byte
If Initialized Then
    XMLWriteProperties strContents
5    b = ChrW$(&HFEFF) & strContents
    Contents = b
Else
    Contents = ""
End If
10 Exit Property
ErrorHandler:
    Contents = Empty
    ErrorMessage Err.Number, "FChType.Contents", UNEXPECTED
End Property
15
*****

Private Sub XMLReadProperties(ByVal xml As String)
*****

    '* Reads the object properties from a XML string
    '* xml contains the XML string
20 *****

    On Error GoTo ErrorHandler
    Dim objDOMDocument As MSXML.DOMDocument
    Dim rootElement As MSXML.IXMLDOMElement
    Dim childElement As MSXML.IXMLDOMElement
25 Dim childElements As MSXML.IXMLDOMNodeList
    Dim strTagName As String
    Dim strAddInfo As String
    Dim ExecutableTypeName As String
    Set objDOMDocument = New MSXML.DOMDocument
30 If objDOMDocument.loadXML(xml) Then
        Set rootElement = objDOMDocument.documentElement

```

```

mExecutableType = XMLReadExecutableType(rootElement)
Set childElements = rootElement.childNodes
For Each childElement In childElements
    strTagName = childElement.TagName
5    Select Case strTagName
        Case "Interface"
            If Not mInterfaceLoaded Then
                mInterface.XMLReadProperties childElement
                mInterfaceLoaded = True
10            End If
        Case "Body"
            If Not mBodyLoaded Then
                mBody.XMLReadProperties childElement
                mBodyLoaded = True
15            End If
        Case Else
            ErrorMessage ERRMOD_XML_TAG,
"FChType.XMLReadProperties", WARNING, strTagName
            End Select
20    Next
Else
    strAddInfo = objDOMDocument.parseError.Reason
    On Error GoTo 0
    ErrorMessage ERRMOD_XML_PARSER,
25 "FChType.XMLReadProperties", ALARM, strAddInfo
    End If
    Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "FChType.XMLReadProperties",
30 UNEXPECTED
End Sub

```

\*\*\*\*\*

**Private Function XMLReadExecutableType(xmlElement As  
MSXML.IXMLDOMElement) As FChExecutableType**

5 \*\*\*\*\*

'\* Reads the ExecutableType property from a XML object

'\* xmlElement contains the XML element

'\* Return value is the ExecutableType

\*\*\*\*\*

10 On Error GoTo ErrorHandler

Dim strExecutabletype As String

strExecutabletype = XMLGetAttribute(xmlElement,  
"ExecutableTypeName")

If strExecutabletype = "FUNCTION\_BLOCK" Then

15 XMLReadExecutableType = FUNCTION\_BLOCK

Else

ErrorMessage ERRMOD\_UNSUPPORTED\_EXECUTABLETYPE,  
"FChType.XMLReadProperties", WARNING, \_  
strExecutabletype

20 End If

Exit Function

ErrorHandler:

XMLReadExecutableType = FUNCTION\_BLOCK

ErrorMessage ERRMOD\_XML\_TAG,

25 "FChType.XMLReadExecutableType", WARNING

End Function

\*\*\*\*\*

30 **Private Sub XMLWriteProperties(xml As String)**

\*\*\*\*\*

'\* Writes the properties to an xml String

\* The xml string is return in the variable xml

\*\*\*\*\*

On Error GoTo ErrorHandler

Dim objDOMDocument As MSXML.DOMDocument

5 Dim rootElement As MSXML.IXMLDOMElement

Dim childElement As MSXML.IXMLDOMElement

Dim NewSize As Long

NewSize = (CLng(mBody.Elements.Count \ 32) + 1) \* 32 'change trace size

mInterface.ChangeTraceSize NewSize

10 Set objDOMDocument = New MSXML.DOMDocument

Set rootElement = objDOMDocument.CreateElement("ChartSource")

Set objDOMDocument.documentElement = rootElement

rootElement.SetAttribute "ExecutableTypeName", ExecutableTypeName

mInterface.XMLWriteProperties rootElement

15 mBody.XMLWriteProperties rootElement

xml = "<?xml version=""1.0"" ?>" & vbCrLf & \_

Replace(objDOMDocument.xml, "><", ">" & vbCrLf & "<")

Exit Sub

ErrorHandler:

20 ErrorMessage Err.Number, "FChType.XMLWriteProperties",

UNEXPECTED

End Sub

25

## Class FChInterface

\*\*\*\*\*

30 Friend Sub XMLReadProperties(xmlElement As

MSXML.IXMLDOMElement)

\*\*\*\*\*

\* Reads the properties of the FChInterface object from a XML object

\*\*\*\*\*

```

On Error GoTo ErrorHandler
Dim childElement As MSXML.IXMLDOMElement
5 Dim childElements As MSXML.IXMLDOMNodeList
Dim strSectionName As String
Dim lngResult As Long
Dim IfSections As S7_Component_Interface_Editor_Server.IfxCollection
CalledByInternal = True
10 Set mIfServer = New S7_Component_Interface_Editor_Server.Interface
If Not mChartType Is Nothing Then
    lngResult = mIfServer.Create(S7_FB_IF)
Elseif Not mChartTask Is Nothing Then
    lngResult = mIfServer.Create(S7_OB_IF)
15 End If
Set mRootNode = mIfServer.RootNode
If mRootNode.HasChildren Then
    Set IfSections = mRootNode.Children
    Set childElements = xmlElement.childNodes
20 For Each childElement In childElements
    If childElement.TagName = "Section" Then
        strSectionName = XMLGetAttribute(childElement, "Name")
        Select Case strSectionName
            Case "VAR_INPUT"
25 XMLCreateSection childElement, IfSections.GetItem("IN")
            Case "VAR_OUTPUT"
                XMLCreateSection childElement, IfSections.GetItem("OUT")
            Case "VAR_IN_OUT"
                XMLCreateSection childElement, IfSections.GetItem("IN_OUT")
30 Case "VAR"
                XMLCreateSection childElement, IfSections.GetItem("STAT")
            Case "VAR_TEMP"
```

```

XMLCreateSection childElement, IfSections.GetItem("TEMP")
Case Else
End Select
Else
5      ErrorMessage ERRMOD_XML_TAG,
      "FChInterface.XMLReadProperties", WARNING, _
      childElement.TagName
End If
Next
10    Debug.Print
Else
      On Error GoTo 0
      ErrorMessage ERRMOD_INTERFACE_CREATION_FAILED,
      "FChInterface.XMLReadProperties", ALARM
15    End If
      CalledByInternal = False
      mModified = False
      Exit Sub
ErrorHandler:
20    CalledByInternal = False
      ErrorMessage Err.Number, "FChInterface.XMLReadProperties",
      UNEXPECTED
End Sub

25
*****

Private Sub XMLCreateSection(xmlElement As
MSXML.IXMLDOMElement, _
      IFItem As
30    S7_Component_Interface_Editor_Server.InterfaceItem)
*****

  * Reads the properties of an Interface section from a XML object

```

\*\*\*\*\*

```

On Error GoTo ErrorHandler
Dim childElement As MSXML.IXMLDOMElement
Dim childElements As MSXML.IXMLDOMNodeList
5 Dim childItem As S7_Component_Interface_Editor_Server.InterfaceItem
Dim lngResult As Long
Dim pvarCorrectnessBar As Variant
Dim strAttrValue As String
Dim xmlAttr As MSXML.IXMLDOMAttribute
10 Dim IChartType As FChType
If IFItem Is Nothing Then GoTo ErrorHandler
Set childElements = xmlElement.childNodes
For Each childElement In childElements
    If childElement.TagName = "Item" Then
15 Set childItem = IFItem.NewChild(-1)
        childItem.ItemProtectionMode = MODE_EDIT
        strAttrValue = XMLGetAttribute(childElement, "Name")
        If Len(strAttrValue) > 0 Then
            lngResult = childItem.SetAttributeString(ATTRIBUTE_NAME,
20 strAttrValue, _
            pvarCorrectnessBar)
            If lngResult <> 0 Then GoTo ErrorHandler
        End If
        strAttrValue = XMLGetAttribute(childElement, "FChType") ' special
25 handling of Subcharts
        If Len(strAttrValue) > 0 Then
            Set IChartType = mChartTypes.Item(strAttrValue)
            If Not IChartType Is Nothing Then
                strAttrValue = IChartType.ExecutableName
30 Set IChartType = Nothing
            End If
        End If
    End If

```

```

    If Len(strAttrValue) = 0 Then
        strAttrValue = XMLGetAttribute(childElement, "Type")
    End If
    If Len(strAttrValue) > 0 Then
5       lngResult = childItem.SetAttributeString(ATTRIBUTE_TYPE,
        strAttrValue, _
        pvarCorrectnessBar)
        If lngResult <> 0 Then GoTo ErrorHandler
    End If
10      strAttrValue = XMLGetAttribute(childElement, "Init")
    If Len(strAttrValue) > 0 Then
        lngResult = childItem.SetAttributeString(ATTRIBUTE_INITIAL,
        strAttrValue, _
        pvarCorrectnessBar)
15      If lngResult <> 0 Then GoTo ErrorHandler
    End If
    XMLCreateSection childElement, childItem
    'must be called before "Mode" is set (if it is ReadOnly ...)
    strAttrValue = XMLGetAttribute(childElement, "Mode")
20    If Len(strAttrValue) > 0 Then
        If IsNumeric(strAttrValue) Then
            childItem.ItemProtectionMode = CLng(strAttrValue)
        ElseIf ItemProtectionModes.Exists(strAttrValue) Then
            childItem.ItemProtectionMode =
25    ItemProtectionModes(strAttrValue)
        End If
    End If
    ElseIf childElement.TagName = "Attributes" Then
        For Each xmlAttr In childElement.Attributes
30      lngResult = IFItem.SetUDA(xmlAttr.Name, xmlAttr.Value)
        If lngResult <> 0 Then GoTo ErrorHandler
    Next

```

```

    ElseIf childElement.TagName = "Comment" Then
        strAttrValue = childElement.Text
        If Len(strAttrValue) > 0 Then
            lngResult = IFItem.SetAttributeString(ATTRIBUTE_COMMENT,
5      strAttrValue, _
            pvarCorrectnessBar)
            If lngResult <> 0 Then GoTo ErrorHandler
        End If
    Else
10      ErrorMessage ERRMOD_XML_TAG,
        "FChInterface.XMLReadProperties", WARNING, _
        childElement.TagName
    End If
    Next
15      Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "FChInterface.XMLCreateSection",
    UNEXPECTED
End Sub

```

20

\*\*\*\*\*

### **Friend Sub XMLWriteProperties(xmlElement As MSXML.IXMLDOMElement)**

25

\*\*\*\*\*

!\* Writes the properties of the FChInterface object to a XML object

\*\*\*\*\*

```

    On Error GoTo ErrorHandler
    Dim childElement As MSXML.IXMLDOMElement
    Dim SectionElement As MSXML.IXMLDOMElement
30      Set childElement =
        xmlElement.ownerDocument.CreateElement("Interface")

```

```
xmlElement.appendChild childElement
```

```
If Not mChartType Is Nothing Then ' these sections do not exist in OBs =
```

```
Tasks
```

```
If mlfServer.InParameter.Count > 0 Then
```

```
5      Set SectionElement =
```

```
childElement.ownerDocument.CreateElement("Section")
```

```
childElement.appendChild SectionElement
```

```
SectionElement.SetAttribute "Name", "VAR_INPUT"
```

```
XMLSectionAsText mlfServer.InParameter, SectionElement
```

```
10    End If
```

```
If mlfServer.OutParameter.Count > 0 Then
```

```
Set SectionElement =
```

```
childElement.ownerDocument.CreateElement("Section")
```

```
childElement.appendChild SectionElement
```

```
15    SectionElement.SetAttribute "Name", "VAR_OUTPUT"
```

```
XMLSectionAsText mlfServer.OutParameter, SectionElement
```

```
End If
```

```
If mlfServer.InOutParameter.Count > 0 Then
```

```
Set SectionElement =
```

```
20    childElement.ownerDocument.CreateElement("Section")
```

```
childElement.appendChild SectionElement
```

```
SectionElement.SetAttribute "Name", "VAR_IN_OUT"
```

```
XMLSectionAsText mlfServer.InOutParameter, SectionElement
```

```
End If
```

```
25    If mlfServer.StaticData.Count > 0 Then
```

```
Set SectionElement =
```

```
childElement.ownerDocument.CreateElement("Section")
```

```
childElement.appendChild SectionElement
```

```
SectionElement.SetAttribute "Name", "VAR"
```

```
30    XMLSectionAsText mlfServer.StaticData, SectionElement
```

```
End If
```

```
End If
```

```

    If mlfServer.DynamicData.Count > 0 Then
        Set SectionElement =
childElement.ownerDocument.CreateElement("Section")
        childElement.appendChild SectionElement
5       SectionElement.SetAttribute "Name", "VAR_TEMP"
        XMLSectionAsText mlfServer.DynamicData, SectionElement
    End If
    Exit Sub
ErrorHandler:
10    ErrorMessage Err.Number, "FChInterface.XMLWriteProperties",
    UNEXPECTED
End Sub

15 *****
Private Sub XMLSectionAsText(Section As
S7_Component_Interface_Editor_Server.IIfxCollection, _
xmlElement As MSXML.IXMLDOMElement)
*****
20  '* Writes the properties of an Interface section to a XML object
*****

    On Error GoTo ErrorHandler
    Dim ItemCount As Long
    Dim Index As Long
25    Dim IFItem As S7_Component_Interface_Editor_Server.InterfaceItem
    ItemCount = Section.Count - 1
    For Index = 0 To ItemCount
        Set IFItem = Section.GetItem(Index)
        XMLItemAsText IFItem, xmlElement
30    Next
    Exit Sub
ErrorHandler:

```

ErrorMessage Err.Number, "FChInterface.XMLSectionAsText",  
 UNEXPECTED  
 End Sub

5

\*\*\*\*\*

**Private Sub XMLItemAsText(IFItem As  
 S7\_Component\_Interface\_Editor\_Server.InterfaceItem, \_  
 xmlElement As MSXML.IXMLDOMElement)**

10

\*\*\*\*\*

'\* Writes the properties of an Interface item to a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

Dim Status As Boolean

15

Dim InitValue As Variant

Dim InitString As String

Dim Pos As Long

Dim udaList As S7\_Component\_Interface\_Editor\_Server.IIfxCollection

Dim udaltem As S7\_Component\_Interface\_Editor\_Server.IUDA

20

Dim i As Long

Dim udaCount As Long

Dim strComment As String

Dim childElement As MSXML.IXMLDOMElement

Dim UDAElement As MSXML.IXMLDOMElement

25

Dim strType As String

Dim strFChType As String

Dim TypeID As S7TypeConstants

Dim TypeInfo As Variant

Dim SubTypeID As S7TypeConstants

30

Dim SubTypeInfo As Variant

Dim Result As Long

Set childElement = xmlElement.ownerDocument.CreateElement("Item")

```

xmlElement.appendChild childElement
childElement.SetAttribute "Name", IFItem.Name
If ItemProtectionModes.Exists(IFItem.ItemProtectionMode) Then
    childElement.SetAttribute "Mode",
5 ItemProtectionModes(IFItem.ItemProtectionMode)
    Else
        childElement.SetAttribute "Mode", IFItem.ItemProtectionMode
    End If
    Set udaList = IFItem.udaList
10    udaCount = udaList.Count
    If udaCount > 0 Then
        Set UDAElement =
xmlElement.ownerDocument.CreateElement("Attributes")
        childElement.appendChild UDAElement
15    For i = 0 To udaCount - 1
        Set udaltem = udaList.GetItem(i)
        UDAElement.SetAttribute udaltem.Key, udaltem.Value
    Next
    End If
20    strType = IFItem.GetAttributeString(ATTRIBUTE_TYPE, Status)
    If Mid$(strType, 1, 3) = "FB " Then strType = Replace(strType, " ", "")
    Result = IFItem.GetTypeInfo(TypeID, TypeInfo, SubTypeID, SubTypeInfo)
    If TypeID = TYPE_S7_TYPE_FB Then
        strFChType = SearchForSubChart(strType)
25    If Len(strFChType) > 0 Then
        childElement.SetAttribute "FChType", strFChType
    End If
    End If
    childElement.SetAttribute "Type", strType
30    If InterfaceItemHasChildren(strType) Then
        XMLSectionAsText IFItem.Children, childElement
    Else

```

```

InitString = GetInitString(IFItem)
If Len(InitString) > 0 Then
    childElement.SetAttribute "Init", InitString
End If
5   End If
XMLWriteTextNode childElement, "Comment", IFItem.Comment
Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "FChInterface.XMLItemAsText",
10  UNEXPECTED
End Sub

```

### **Class FChBody**

```

15  *****

Friend Sub XMLReadProperties(xmlElement As
MSXML.IXMLDOMElement)
    *****

20  '* Reads the properties of a FChBody object from a XML object
    *****

    On Error GoTo ErrorHandler
    Dim childElement As MSXML.IXMLDOMElement
    Dim childElements As MSXML.IXMLDOMNodeList
25  Set mElements = New FChElements
    Dim objFChElement As FChElement
    Set mElements.Body = Me
    Set mLinks = New FChLinks
    Set mLinks.Body = Me
30  Set childElements = xmlElement.childNodes
    For Each childElement In childElements
        Select Case childElement.TagName

```

```

Case "Elements"
    mElements.XMLReadProperties childElement
Case "Links"
    mLinks.XMLReadProperties childElement
5   Case Else
        ErrorMessage ERRMOD_XML_TAG,
"FChBody.XMLReadProperties", WARNING, childElement.TagName
    End Select
Next
10  Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "FChBody.XMLReadProperties",
UNEXPECTED
End Sub
15

*****

Friend Sub XMLWriteProperties(xmlElement As
MSXML.IXMLDOMElement)
20  *****

    '* Writes the properties of FChBody object to a XML object
    *****

    On Error GoTo ErrorHandler
    Dim childElement As MSXML.IXMLDOMElement
25  Set childElement = xmlElement.ownerDocument.CreateElement("Body")
    xmlElement.appendChild childElement
    mElements.XMLWriteProperties childElement
    mLinks.XMLWriteProperties childElement
    Exit Sub
30  ErrorHandler:
        ErrorMessage Err.Number, "FChBody.XMLWriteProperties",
UNEXPECTED

```

End Sub

\*\*\*\*\*

## 5 Class FChElements

\*\*\*\*\*

**Friend Sub XMLReadProperties(xmlElement As  
MSXML.IXMLDOMElement)**

\*\*\*\*\*

10 '\* Reads the properties of a FChElements object from a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

Dim objFChElement As FChElement

Dim childElement As MSXML.IXMLDOMElement

15 Dim childElements As MSXML.IXMLDOMNodeList

Set childElements = xmlElement.childNodes

mIndex = 0

For Each childElement In childElements

If childElement.TagName = "Element" Then

20 Set objFChElement = New FChElement

Set objFChElement.Body = mBody

objFChElement.XMLReadProperties childElement

If mIndex <= objFChElement.Number Then mIndex =

objFChElement.Number + 1

25 mBody.RaiseCreatedEvent objFChElement

mCol.Add objFChElement, Format\$(objFChElement.Number)

Else

ErrorMessage ERRMOD\_XML\_TAG,

"FChElements.XMLReadProperties", WARNING, \_

30 childElement.TagName

End If

Next

Exit Sub

ErrorHandler:

ErrorMessage Err.Number, "FChElements.XMLReadProperties",  
UNEXPECTED

5 End Sub

\*\*\*\*\*

**Friend Sub XMLWriteProperties(xmlElement As  
10 MSXML.IXMLDOMElement)**

\*\*\*\*\*

'\* Writes the properties of FChElements object to a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

15 Dim objFChElement As FChElement

Dim childElement As MSXML.IXMLDOMElement

Set childElement =

xmlElement.ownerDocument.CreateElement("Elements")

xmlElement.appendChild childElement

20 For Each objFChElement In mCol

objFChElement.XMLWriteProperties childElement

Next

Exit Sub

ErrorHandler:

25 ErrorMessage Err.Number, "FChElements.XMLWriteProperties",  
UNEXPECTED

End Sub

30 \*\*\*\*\*

**Class FChLinks**

\*\*\*\*\*

**Friend Sub XMLReadProperties(xmlElement As  
MSXML.IXMLDOMElement)**

\*\*\*\*\*

\* Reads the properties of a FChLinks object from a XML object

\*\*\*\*\*

```

5  On Error GoTo ErrorHandler
    Dim objFChlink As FChLink
    Dim childElement As MSXML.IXMLDOMElement
    Dim childElements As MSXML.IXMLDOMNodeList
10  Set childElements = xmlElement.childNodes
    For Each childElement In childElements
        If childElement.TagName = "Link" Then
            Set objFChlink = New FChLink
            Set objFChlink.Body = mBody
15  objFChlink.XMLReadProperties childElement
            If mIndex <= objFChlink.Number Then mIndex = objFChlink.Number +
20  1
                mBody.RaiseCreatedEvent objFChlink
                mCol.Add objFChlink, Format$(objFChlink.Number)
                objFChlink.UpdateLinksAdd
            Else
                ErrorMessage ERRMOD_XML_TAG,
                "FChLinks.XMLReadProperties", WARNING, childElement.TagName
            End If
25  Next
    Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "FChLinks.XMLReadProperties",
    UNEXPECTED
30 End Sub

```

\*\*\*\*\*

**Friend Sub XMLWriteProperties(xmlElement As  
MSXML.IXMLDOMElement)**

\*\*\*\*\*

'\* Writes the properties of FChLinks object to a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

10 Dim objFChlink As FChLink

Dim childElement As MSXML.IXMLDOMElement

Set childElement = xmlElement.ownerDocument.CreateElement("Links")

xmlElement.appendChild childElement

For Each objFChlink In mCol

15 objFChlink.XMLWriteProperties childElement

Next

Exit Sub

ErrorHandler:

ErrorMessage Err.Number, "FChLinks.XMLWriteProperties",

20 UNEXPECTED

End Sub

\*\*\*\*\*

25 **Class FChElement**

\*\*\*\*\*

**Friend Sub XMLReadProperties(xmlElement As  
MSXML.IXMLDOMElement)**

\*\*\*\*\*

30 '\* Reads the properties of a FChElement object from a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

```

Dim tmpElementType As String
Dim lngIndex As Long
Dim childElement As MSXML.IXMLDOMElement
Dim childElements As MSXML.IXMLDOMNodeList
5 Dim strOutLinks As String
Dim strCommentPosition As String
Dim arrayCommentPosition() As String
tmpElementType = XMLGetAttribute(xmlElement, "Type")
If Not ElementTypeDic.Exists(tmpElementType) Then
10 On Error GoTo 0
    ErrorMessage ERRMOD_UNKOWN_ELEMENT_TYPE,
    "FChElement.XMLReadProperties", ALARM, _
    tmpElementType
    Else
15 ElementType = ElementTypeDic(tmpElementType)
    mNumber = XMLGetAttribute(xmlElement, "Number")
    mCaption = XMLGetAttribute(xmlElement, "Caption")
    mDefaultCaption = (mCaption = "%D%")
    If mElementType = TGoto Then
20 mGotoTargetNumber = XMLGetAttribute(xmlElement,
    "GotoTargetNumber")
    End If
    If mElementType = TComment Then
        strCommentPosition = XMLGetAttribute(xmlElement,
25 "CommentPosition")
        If Len(strCommentPosition) > 0 Then
            arrayCommentPosition = Split(strCommentPosition, ",")
            If Not IsEmpty(arrayCommentPosition) Then
                If UBound(arrayCommentPosition) = 3 Then
30 For lngIndex = 0 To 3
                    mCommentPosition(lngIndex) =
                    CLng(arrayCommentPosition(lngIndex))

```

```

        Next
    End If
End If
End If
5   End If
    Set childElements = xmlElement.childNodes
    For Each childElement In childElements
        Select Case childElement.TagName
            Case "SourceCode"
10          If (mElementType = TAction) Or (mElementType = TDecision) Then
                mSourceCode = childElement.Text
            Else
                ErrorMessage ERRMOD_XML_TAG,
                "FChElement.XMLReadProperties", WARNING, _
15          childElement.TagName
            End If
            Case "Comment"
                mComment = childElement.Text
            Case "Instance"
20          If mElementType = TSubChart Then
                Set mInstance = New FChInstance
                Set mInstance.Body = mBody
                Set mInstance.Element = Me
                mInstance.XMLReadProperties childElement
25          Else
                ErrorMessage ERRMOD_XML_TAG,
                "FChElement.XMLReadProperties", WARNING, _
                childElement.TagName
            End If
30          Case Else
                ErrorMessage ERRMOD_XML_TAG,
                "FChElement.XMLReadProperties", WARNING, _

```

childElement.TagName

End Select

Next

End If

5 Exit Sub

ErrorHandler:

ErrorMessage Err.Number, "FChElement.XMLReadProperties",

UNEXPECTED

End Sub

10

\*\*\*\*\*

**Friend Sub XMLWriteProperties(xmlElement As  
MSXML.IXMLDOMElement)**

15

\*\*\*\*\*

'\* Writes the properties of FChElement object to a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

Dim childElement As MSXML.IXMLDOMElement

20

Set childElement = xmlElement.ownerDocument.CreateElement("Element")

xmlElement.appendChild childElement

childElement.SetAttribute "Number", mNumber

childElement.SetAttribute "Type", ElementTypeNames(mElementType)

childElement.SetAttribute "Caption", mCaption

25

If mElementType = TGoto Then

childElement.SetAttribute "GotoTargetNumber", mGotoTargetNumber

End If

XMLWriteTextNode childElement, "Comment", mComment

If (mElementType = TAction) Or (mElementType = TDecision) Then

30

XMLWriteTextNode childElement, "SourceCode", mSourceCode

End If

If mElementType = TComment Then

```

        childElement.SetAttribute "CommentPosition",
Str$(mCommentPosition(0)) & "," & _
                                Str$(mCommentPosition(1)) & "," & _
                                Str$(mCommentPosition(2)) & "," & _
5                                Str$(mCommentPosition(3))

```

```
End If
```

```
If Not mInstance Is Nothing Then
```

```
    mInstance.XMLWriteProperties childElement
```

```
End If
```

```
10 Exit Sub
```

```
ErrorHandler:
```

```
    ErrorMessage Err.Number, "FChElement.XMLWriteProperties",
```

```
UNEXPECTED
```

```
End Sub
```

```
15
```

```
*****
```

### **Class FChLink**

```
20
```

```
*****
```

```
Friend Sub XMLReadProperties(xmlElement As  
MSXML.IXMLDOMElement)
```

```
*****
```

```
'* Reads the properties of a FChLink object from a XML object
```

```
25
```

```
*****
```

```
On Error GoTo ErrorHandler
```

```
mNumber = XMLGetAttribute(xmlElement, "Number")
```

```
mSourceNumber = XMLGetAttribute(xmlElement, "SourceElement")
```

```
mTargetNumber = XMLGetAttribute(xmlElement, "TargetElement")
```

```
30
```

```
mIndex = XMLGetAttribute(xmlElement, "Index")
```

```
mCaption = XMLGetAttribute(xmlElement, "Caption")
```

```
mDefaultCaption = (mCaption = "%D%")
```

Exit Sub

ErrorHandler:

ErrorMessage Err.Number, "FChLink.XMLReadProperties", UNEXPECTED

End Sub

5

\*\*\*\*\*

**Friend Sub XMLWriteProperties(xmlElement As  
MSXML.IXMLDOMElement)**

10

\*\*\*\*\*

'\* Writes the properties of FChLink object to a XML object

\*\*\*\*\*

On Error GoTo ErrorHandler

Dim childElement As MSXML.IXMLDOMElement

15

Dim objFChlink As FChLink

Set childElement = xmlElement.ownerDocument.CreateElement("Link")

xmlElement.appendChild childElement

childElement.SetAttribute "Number", mNumber

childElement.SetAttribute "SourceElement", mSourceNumber

20

childElement.SetAttribute "TargetElement", mTargetNumber

childElement.SetAttribute "Index", mIndex

childElement.SetAttribute "Caption", mCaption

Exit Sub

ErrorHandler:

25

ErrorMessage Err.Number, "FChLink.XMLWriteProperties", UNEXPECTED

End Sub

\*\*\*\*\*

30

**Class FChInstance**

\*\*\*\*\*

**Friend Sub XMLReadProperties(xmlElement As**

**MSXML.IXMLDOMElement)**

\*\*\*\*\*

\* Reads the properties of a FChInstance object from a XML object

\*\*\*\*\*

```

5      On Error GoTo ErrorHandler
      Dim childElement As MSXML.IXMLDOMElement
      Dim childElements As MSXML.IXMLDOMNodeList
      Dim mAssignment As FChAssignment
      Dim mSubChartType As FChType
10     Dim strIfVersion As String
      mChartType = XMLGetAttribute(xmlElement, "ChartType")
      mName = XMLGetAttribute(xmlElement, "Name")
      strIfVersion = XMLGetAttribute(xmlElement, "InterfaceVersion")
      If IsNumeric(strIfVersion) And Len(strIfVersion) > 0 Then
15         mInterfaceVersion = Format$(strIfVersion)
      Else
          mInterfaceVersion = 0
      End If
      Set mSubChartType = ChartType
20     Set childElements = xmlElement.childNodes
      For Each childElement In childElements
          Select Case childElement.TagName
              Case "Assignment"
                  Set mAssignment = New FChAssignment
25                 mAssignment.XMLReadProperties childElement
                  If Not mSubChartType Is Nothing Then
                      Set mAssignment.IfServer = mSubChartType.Interface.IfServer
                  End If
                  If AssignmentExists(mAssignment.Name) Then
30                     Set mAssignment = Nothing
                      ErrorMessage ERRMOD_DUPLICATE_ASSIGNMENT,
                          "FChInstance.XMLReadProperties", _

```

WARNING, mAssignment.Name

Else

mCol.Add mAssignment, mAssignment.Name

End If

5 Set mAssignment = Nothing

Case Else

ErrorMessage ERRMOD\_XML\_TAG,

"FChInstance.XMLReadProperties", WARNING, \_  
childElement.TagName

10 End Select

Next

Set mSubChartType = Nothing

Exit Sub

ErrorHandler:

15 ErrorMessage Err.Number, "FChInstance.XMLReadProperties",

UNEXPECTED

End Sub

20 '\*\*\*\*\*

**Friend Sub XMLWriteProperties(xmlElement As  
MSXML.IXMLDOMElement)**

'\*\*\*\*\*

'\* Writes the properties of FChInstance object to a XML object

25 '\*\*\*\*\*

On Error GoTo ErrorHandler

Dim childElement As MSXML.IXMLDOMElement

Dim mAssignment As FChAssignment

Set childElement =

30 xmlElement.ownerDocument.CreateElement("Instance")

xmlElement.appendChild childElement

childElement.SetAttribute "Name", mName

```

childElement.SetAttribute "InterfaceVersion", mInterfaceVersion
childElement.SetAttribute "ChartType", mChartType
For Each mAssignment In mCol
    mAssignment.XMLWriteProperties childElement
5    Next
    Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "FChInstance.XMLWriteProperties",
    UNEXPECTED
10 End Sub

'*****

Class FChAssignment
15 '*****

Friend Sub XMLReadProperties(xmlElement As
MSXML.IXMLDOMElement)
'*****

' * Reads the properties of a FChAssignment object from a XML object
20 '*****

    On Error GoTo ErrorHandler
    mName = XMLGetAttribute(xmlElement, "Name")
    mValue = XMLGetAttribute(xmlElement, "Value")
    Exit Sub
25 ErrorHandler:
    ErrorMessage Err.Number, "FChAssignment.XMLReadProperties",
    UNEXPECTED
    End Sub

30 '*****

Friend Sub XMLWriteProperties(xmlElement As

```

**MSXML.IXMLDOMElement)**

\*\*\*\*\*

'\* Writes the properties of FChAssignment object to a XML object

\*\*\*\*\*

```

5      On Error GoTo ErrorHandler
      Dim childElement As MSXML.IXMLDOMElement
      Dim mAssignment As FChAssignment
      Set childElement =
xmlElement.ownerDocument.CreateElement("Assignment")
10     xmlElement.appendChild childElement
        childElement.SetAttribute "Name", mName
        childElement.SetAttribute "Value", mValue
        Exit Sub
ErrorHandler:
15     ErrorMessage Err.Number, "FChAssignment.XMLWriteProperties",
        UNEXPECTED
    End Sub

```

20

\*\*\*\*\*

**Global Subroutines and Functions**

\*\*\*\*\*

```

Public Sub XMLWriteTextNode(xmlElement As
25 MSXML.IXMLDOMElement, Name As String, Data As String)

```

\*\*\*\*\*

'\* Writes a property as text node to a XML object

'\* Property name is Name, property Data is in Data

\*\*\*\*\*

```

30     On Error GoTo ErrorHandler
        Dim childElement As MSXML.IXMLDOMElement
        Dim TextElement As MSXML.IXMLDOMText

```

```

If Len(Data) > 0 Then
    Set childElement = xmlElement.ownerDocument.CreateElement(Name)
    xmlElement.appendChild childElement
    Set TextElement = xmlElement.ownerDocument.createTextNode(Data)
5    childElement.appendChild TextElement
End If
Exit Sub
ErrorHandler:
    ErrorMessage Err.Number, "global.XMLWriteTextNode", UNEXPECTED
10 End Sub

*****

Public Function XMLCreateRootElement(ByVal TagName As String) As
15 MSXML.IXMLDOMElement
*****

'* Creates an XML root element with the specified TagName
*****

    On Error GoTo ErrorHandler
20    Dim objDOMDocument As MSXML.DOMDocument
    Dim rootElement As MSXML.IXMLDOMElement
    Set objDOMDocument = New MSXML.DOMDocument
    Set rootElement = objDOMDocument.CreateElement(TagName)
    Set objDOMDocument.documentElement = rootElement
25    Set XMLCreateRootElement = rootElement
    Exit Function
ErrorHandler:
    ErrorMessage Err.Number, "global.XMLCreateRootElement",
    UNEXPECTED
30 End Function

```

\*\*\*\*\*

**Public Function XMLCreateDocument(ByVal Contents As Variant) As  
MSXML.DOMDocument**

\*\*\*\*\*

5 '\* Creates an XML document with the specified xml Contents

\*\*\*\*\*

```

    On Error GoTo ErrorHandler
    Dim xml As String
    Dim objDOMDocument As MSXML.DOMDocument
10  Dim rootElement As MSXML.IXMLDOMElement
    Set objDOMDocument = New MSXML.DOMDocument
    xml = Contents
    If objDOMDocument.loadXML(xml) Then
        Set XMLCreateDocument = objDOMDocument
15  Else
        Set XMLCreateDocument = Nothing
        On Error GoTo 0
        ErrorMessage ERRMOD_XML_PARSER,
"global.XMLCreateDocument", ALARM
20  End If
    Exit Function
ErrorHandler:
    ErrorMessage Err.Number, "global.XMLCreateDocument", UNEXPECTED
End Function

```

25

\*\*\*\*\*

**Public Function XMLGetAttribute(xmlElement As  
MSXML.IXMLDOMElement, Name As String) As String**

30 \*\*\*\*\*

'\* Gets an Attribute from an XML element (needed because of error handling)

\*\*\*\*\*

On Error GoTo ErrorHandler

XMLGetAttribute = xmlElement.GetAttribute(Name)

Exit Function

ErrorHandler:

5 XMLGetAttribute = ""

End Function

\*\*\*\*\*

10 **Public Sub ErrorMessage(ByVal Number As Long, \_  
ByVal Source As String, \_  
ByVal Severity As ErrorSeverity, \_  
Optional ByVal AddInfo As String = "")**

\*\*\*\*\*

15 **'\* Logs an Error message and raises an Error**

\*\*\*\*\*

Dim strTmp As String

Dim lngErrorNumber As Long

Dim lngHelpContextID As Long

20 Select Case Severity

Case WARNING ' Logging only, no error raised

strTmp = objResourceAccess.GetResString(Number, "Error")

If Len(AddInfo) > 0 Then strTmp = strTmp & ": " & AddInfo

lngErrorNumber = vbObjectError + ERRMOD\_BASE\_NUMBER +

25 Number

lngHelpContextID = HELP\_ON\_ERRMOD\_BASE\_NUMBER +

Number

PrintForDebug TraceFileName, strTmp & ": " & Source

Case ALARM ' Error is logged and Error is raised

30 strTmp = objResourceAccess.GetResString(Number, "Error")

If Len(AddInfo) > 0 Then strTmp = strTmp & ": " & AddInfo

lngErrorNumber = vbObjectError + ERRMOD\_BASE\_NUMBER +

Number

IngHelpContextID = HELP\_ON\_ERRMOD\_BASE\_NUMBER +

Number

PrintForDebug TraceFileName, strTmp & ":" & Source

5 Err.Raise lngErrorNumber, Source, strTmp, App.HelpFile,  
IngHelpContextID

Case INTERNAL ' General "internal error" is raised, but detailed logged

strTmp = objResourceAccess.GetResString(Number, "Error")

If Len(AddInfo) > 0 Then strTmp = strTmp & ": " & AddInfo

10 lngErrorNumber = vbObjectError + ERRMOD\_BASE\_NUMBER +  
Number

IngHelpContextID = HELP\_ON\_ERRMOD\_BASE\_NUMBER +

Number

PrintForDebug TraceFileName, strTmp & ":" & Source

15 strTmp = objResourceAccess.GetResString(ERRMOD\_INTERNAL,  
"Error")

lngErrorNumber = vbObjectError + ERRMOD\_BASE\_NUMBER +  
ERRMOD\_INTERNAL

20 lngHelpContextID = HELP\_ON\_ERRMOD\_BASE\_NUMBER +  
ERRMOD\_INTERNAL

Err.Raise lngErrorNumber, Source, strTmp, App.HelpFile,  
IngHelpContextID

Case UNEXPECTED ' Unknown error caused by VB or a subcomponent.  
Treated like warning

25 PrintForDebug TraceFileName, Err.Description & ":" & Source  
Err.Raise Err.Number, Source, Err.Description, Err.HelpFile,  
Err.HelpContext

Case CREATEFILE

PrintForDebug TraceFileName, "START", True

30 Case STEP7ERROR

lngErrorNumber = vbObjectError + ERRMOD\_BASE\_NUMBER +  
Number

\*\*\*\*\*

[illegible]